محمد امانلو تکلیف ششم درس پایگاه داده، استاد شاکری 810100084

سوال اول)

اگر ترتیب record data ها یکسان یا نزدیک به ترتیب entry data ها باشد به آن شاخص clustered میگوییم. اگر چنین ترتیبی وجود نداشته باشد شاخص unclustered است. از آنجا که داده ها با کلید جست و جوی یکسان یا نزدیک در حالت clustered به هم نزدیک ترند ممکن است بر روی صفحات یکسانی قرار گرفته باشند و این امر می تواند باعث کاهش هزینه ها در جست و جوی بازه ای شود. همچنین هر فایل برای حداکثر یک کلید جستجو میتواند clustered باشد. هزینه بازیابی رکوردهای داده ای از طریق شاخص بسیار به clustered بودن یا نبودن بستگی دارد. برای مثال در صورت clustered بودن یک search range را در نظر بگیرید؛ اگر ترتیب record data ها و entry data ها یکی باشد، تنها نیاز است تا record data متناظر با اولین entry data را پیدا کنیم. در حالی که اگر clustered نبود میبایست به ازای همه entry data هایی که شرایط جستوجوی ما داشتند record data متناظر را پیدا میکردیم. در یک فایل نهایتاً روی یک key search میتواند clustered باشد.

الف) بهتر است از شاخص clustered استفاده کنیم. اما در این حالت تفاوت زیاد نیست چرا که در شاخص های درختی از نوع alt2 برای equality search از جنس clustered ما $log\ 1.5\ B\ *\ D$ عملیات IO نیاز است اما در حالت دیگر $log\ (1.5\ B\ +\ 1)\ *\ D$ عملیات نیاز است که تفاوت محسوسی ایجاد نمی کند.

ب) بهتر است که به صورت clustered و ایندکس باشد <محل تولد، سن> چراکه سن های یکسان در کنار هم جای میگیرند و به همین دلیل range search سریعتر و راحت تر است. در واقع به دلیل آنکه مرتبه کلاسترد بسیار بهتر است در این حالت ما این راهکار را پیش میگیریم.

سوال دوم) 1. شاخص هش: در این روش entry data ها را بر روي كليد جست و جو هش میکنیم. در واقع هش یک روش است که در آن داده ها بر اساس یک کلید خاص هش می‌شوند. این به ما این امکان را می‌دهد که داده هایی که یک مقدار خاص از کلید را دارند، به سرعت پیدا کنیم. در این حالت ديتاهايي كه يك مقدار كليد خاص دارند را به سرعت مي توانيم

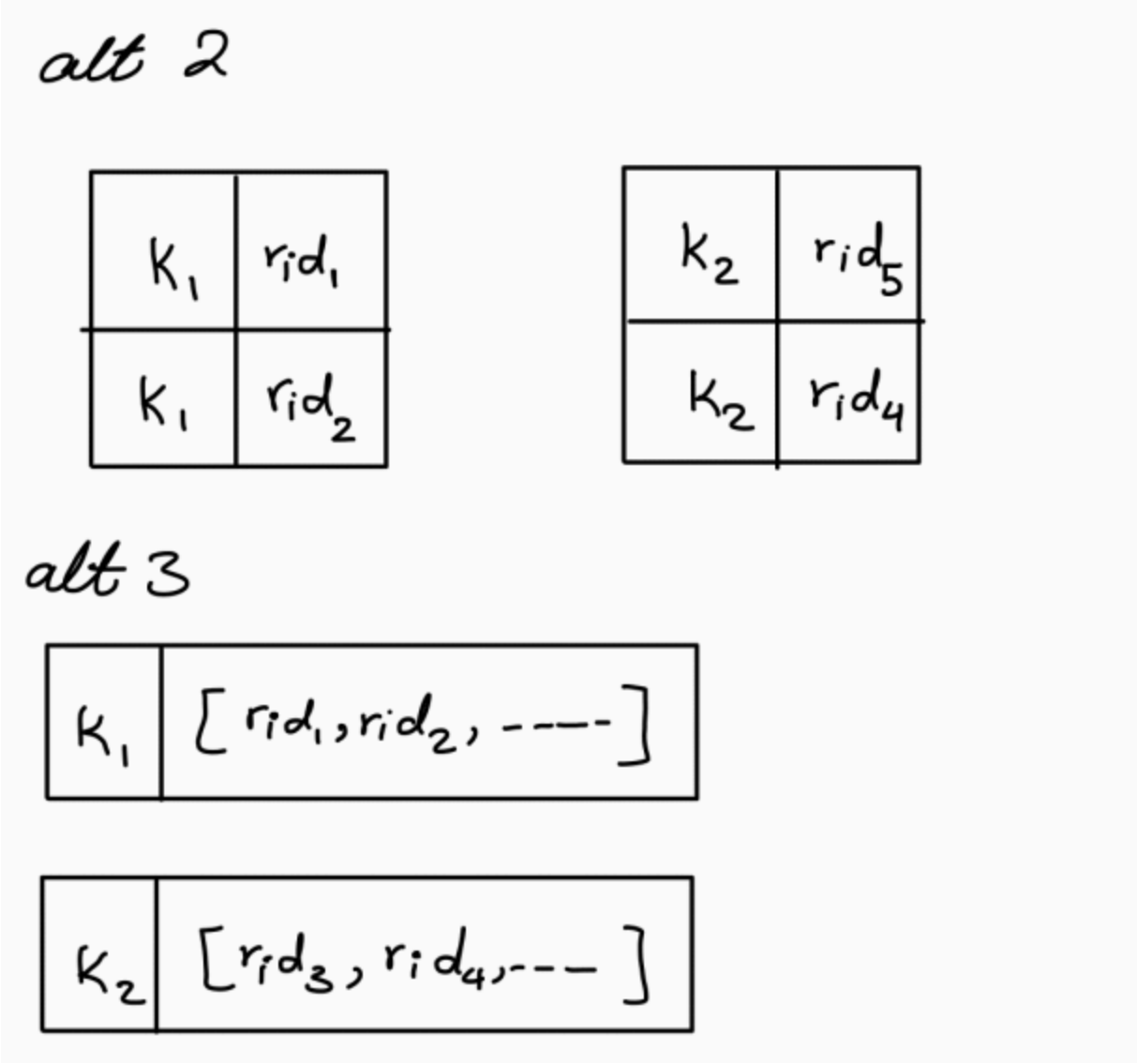
پیدا کنیم. در این روش، هر ایندکس یک مجموعه باکت داده است که روی آن‌ها یک تابع هش اعمال شده است. این تابع هش مشخص می‌کند که داده مربوط به کدام باکت است. هر باکت یک صفحه اصلی دارد و ممکن است یک یا چند صفحه اضافه داشته باشد. این نوع شاخص براي search equality مناسب است چون داده ها با مقدار یکسان کلید جست و جو کنار هم قرار میگیرند و براي search range مناسب نمیباشد.

2. شاخص با Alternative 2 در واقع به این صورت عمل می‌کند که برای هر رکورد، کلید جستجوی آن را همراه با یک اشاره گر به آن رکورد ذخیره می‌کند. اما در شاخص با Alternative 3، برای مقادیر مختلف کلید جستجو، یک جفت از آن مقدار کلید همراه با یک لیست از اشاره‌گرهای رکوردهای دارای آن مقدار کلید جستجو نگهداری می‌شود. در واقع در alt 2 هر data entry شامل search key و rid است.

شاخص با Alternative 3 از نظر فشرده تر بودن و بهره وری فضا بهتر عمل می‌کند اما طول داده های ورودی آن برای هر مقدار کلید ممکن است متفاوت باشد؛ زیرا تعداد رکوردهای مربوط به هر مقدار کلید لزوما برابر نیست. این تفاوت ممکن است بسیار زیاد باشد و باعث عدم تعادل در تعداد رکوردها شود. در واقع در alt 3 در هر سرچ کلید شامل لیست هایی از rid است که شامل آن search key می شوند.

برای مثال، اگر یک شاخص بر اساس سن دانشجویان یک دانشگاه داشته باشیم، احتمالا طول داده‌های ورودی مربوط به مقدار 20 سال (Age = 20) بسیار بیشتر از طول داده‌های مربوط به مقدار 40 سال (Age = 40) خواهد بود. این وضعیت باعث افزایش تفاوت در طول داده‌ها و در نتیجه آثار منفی بر روی کارایی شاخص می‌تواند داشته باشد.

حداکثر یک شاخص تنها می‌تواند Alternative 1 داشته باشد، در غیر این صورت، داده‌های رکورد به صورت تکراری نگهداری می‌شوند که منجر به افزایش ردوندی و ناهماهنگی داده ها می‌شود.



3. بله

```
UPDATE Employee

SET salary = 20000

WHERE  age < 30 AND age > 18 AND ename = "John Doe";
```

در این مثال، ما قصد داریم یک کوئری UPDATE انجام دهیم که بر روی جدول Employee اعمال شود. در این کوئری، ما قصد داریم حقوق (salary) را برای کارمندی به نام "John Doe" که سن او بین 18 تا 30 سال است، به مقدار 20000 تغییر دهیم.

از آنجایی که شرایط جستجوی ما بر اساس سن و نام کارمند مشخص شده است و شاخص‌ها بر روی این فیلدها اعمال شده‌اند، نمی‌توانیم از شاخص‌ها برای این کوئری استفاده کنیم و باید مستقیماً به جستجوی داده‌ها در جدول مراجعه کنیم. این عمل باعث می‌شود که نه تنها از اهمیت و کارایی شاخص‌ها در این عملیات استفاده نشود، بلکه برای اعمال تغییرات باید به صورت مستقیم داده‌های جدول را آپدیت کنیم.

سوال سوم) 1. در روش لیست پیوندی دوطرفه، از دو لیست پیوندی استفاده می‌شود؛ یکی برای صفحات پر و دیگری برای صفحات خالی. وقتی یک داده از یک صفحه پر حذف می‌شود و صفحه جای خالی دارد، آن داده به لیست صفحات خالی منتقل می‌شود، و هرگاه صفحه در حال نوشتن پر شود، به لیست صفحات پر منتقل می‌شود. این روش امکان یافتن داده‌ها و صفحات خالی را به سرعت و با سهولت فراهم می‌آورد. با این حال، اگر طول رکوردها ثابت نباشد، این می‌تواند منجر به ایجاد فضای خالی در انتهای صفحات شود و باعث شود تمام صفحات در لیست صفحات خالی قرار گیرند.

در روش صفحه دایرکتوری، تعدادی دایرکتوری به صورت لیست پیوندی قرار می‌گیرند و در هرکدام اطلاعات مربوط به صفحات (اشاره گر به صفحه و وضعیت پر یا خالی بودن) نوشته می‌شود. این روش به ما این امکان را می‌دهد که میزان فضای خالی را برای هر صفحه نگه‌داریم در این روش، اگر طول رکوردها ثابت نباشد، یک مکان کوچک در انتهای صفحات نمونه گیری می‌شود و عملاً همه چیز در صفحات خالی است. به تعداد زیادی dir داریم که آنها را به صورت linked list قرار می‌دهیم و در سر dir، اطلاعات مربوط به صفحات را می‌نویسیم. دیویتر به صفحه ای که جا دارد یا نه نگاه می‌کند (معمولاً به سر صفحه) و یک مکان خالی در آن پیدا می‌کند. با استفاده از این روش مشکل رکوردها با طول متغیر را حل کنیم.

2. الف) page dir , 10 ptrs در بدترین حالت آدرس صفحه ای که به اندازه کافی جای خالی داشته باشد وجود دارد پس 18 عملیات نیاز است و 2 عملیات هم که برای به روز رسانی و یک عملیات هم برای انتقال به دیسک پس در مجموع 21 عملیات IO لازم است.

ب) در این حالت برای آوردن صفحه به حافظه یک عملیات، در بدترین حالت نیز در ششمین صفحه باید برویم. پس 6 عملیات و بعد از نوشتن و پر شدن باید به پرها منتقل شود پس 1

عملیات سپس پوینتر صفحه قبل را نال میکنیم و در دیسک مینویسیم یک عملیات دیگر. اولین صفحه از لیست پر ها را باید به حافظه بیاوریم 1 عملیات. آپدیت کردن ها نیز 3 عملیات نهایی است پس مجموعا 12 عملیات لازم است.

سوال چهارم) 1. هرگاه اطلاعات موجود در یک شاخص برای پاسخ دادن به یک پرس و جو کافی باشد و نیاز به مراجعه و بازیابی اصل رکوردها نباشد، DBMS یک استراتژی only-index انتخاب میکند و رکوردهای اصلی را بازیابی نمیکند. در واقع Covering index یک تکنیک بهینه سازی است که به عنوان "index-only index" هم شناخته می‌شود. این طرح مربوط به زمانی است که DBMS یک پرس‌وجو را تنها با استفاده از یک ایندکس پاسخ می‌دهد، بدون نیاز به دسترسی به جدول اصلی داده سها. به طور دقیق‌تر، ایندکس تمام اطلاعات لازم برای پاسخ دادن به پرس‌وجو را دارد، به این معنی که DBMS می‌تواند از ایندکس استفاده کند تا پرس‌وجو را خیلی سریعتر انجام دهد بدون نیاز به دسترسی به داده‌های اصلی در جدول. این روش بهینه‌سازی به علت کاهش نیاز به دسترسی به داده‌های اصلی، زمان اجرای پرس‌وجو را بهبود می‌بخشد و کارایی عملیات جستجو را افزایش می‌دهد. در حالت only-index اطلاعات موجود در entry data ها برای پاسخ دادن به کوئریها کافیست و نیازی به واکشی خود داده ها از حافظه نیست. استفاده از این روش باعث کاهش هزینه پرس وجوها میشود.

a.2. شاخص باید به شکل

< category, rate,price>

باشد. در این حالت، داده ها ابتدا بر اساس دسته بندی (category) مرتب می‌شوند و سپس آن‌هایی که دارای همان دسته بندی هستند، بر اساس امتیاز (rate) مرتب می‌شوند. به این ترتیب، می‌توان به راحتی داده هایی که امتیاز بالاتر از 5 دارند را حذف کرد. زیرا در نهایت، کمترین قیمتی که مورد نیاز است را می‌خواهیم در نتیجه، الزامی است که قیمت (price) نیز به عنوان یک شاخص در نظر گرفته شود.

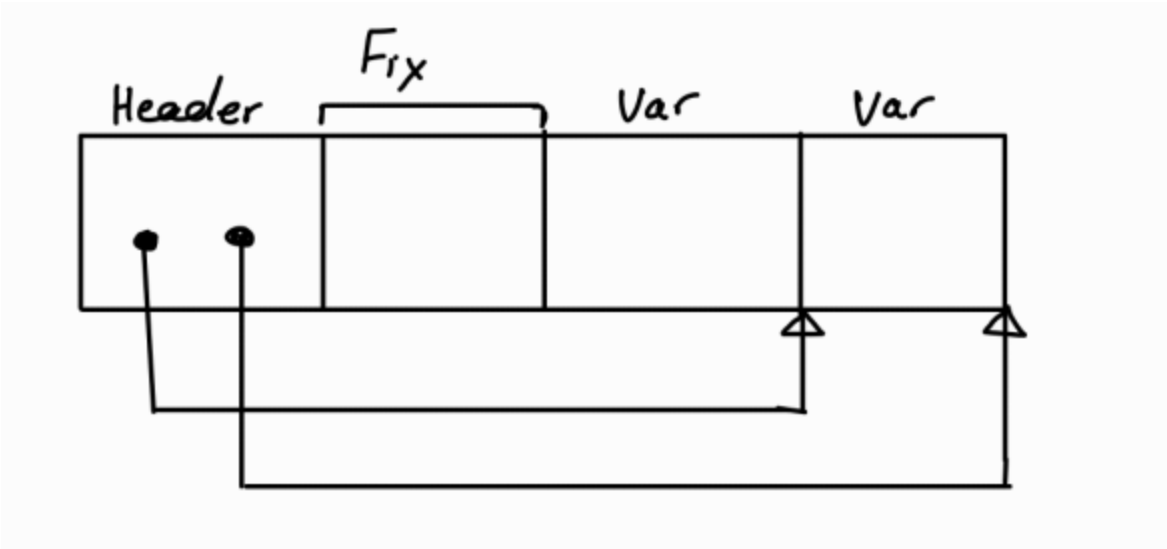
B. در این بخش باید شاخص باید به شکل < category , price> باشد. در واقع اول باید بر اساس کتگوری مرتب بشوند و سپس categoriy های یکسان در کنار هم قرار گرفته تا میانگین قیمت را حساب کنیم.

3. وقتی همه ستون های مورد نیاز ما عضو ایندکس ها باشند این اتفاق می افتد. اما اگر یک ستون مورد نیاز در ایندکس ها نباشد ما امکان ارائه index-only-plan را نداریم. همچنین در صورتی که کوئری های پیچیده با جوین ها و ... داشته باشیم.

Select S.rate
From Suppliers S
Where S.name = “MMD”

در اینجا در صورتی که ایندکس فقط روی نام یا امتیاز باشد استفاده از index-only-plan ممکن نیست.

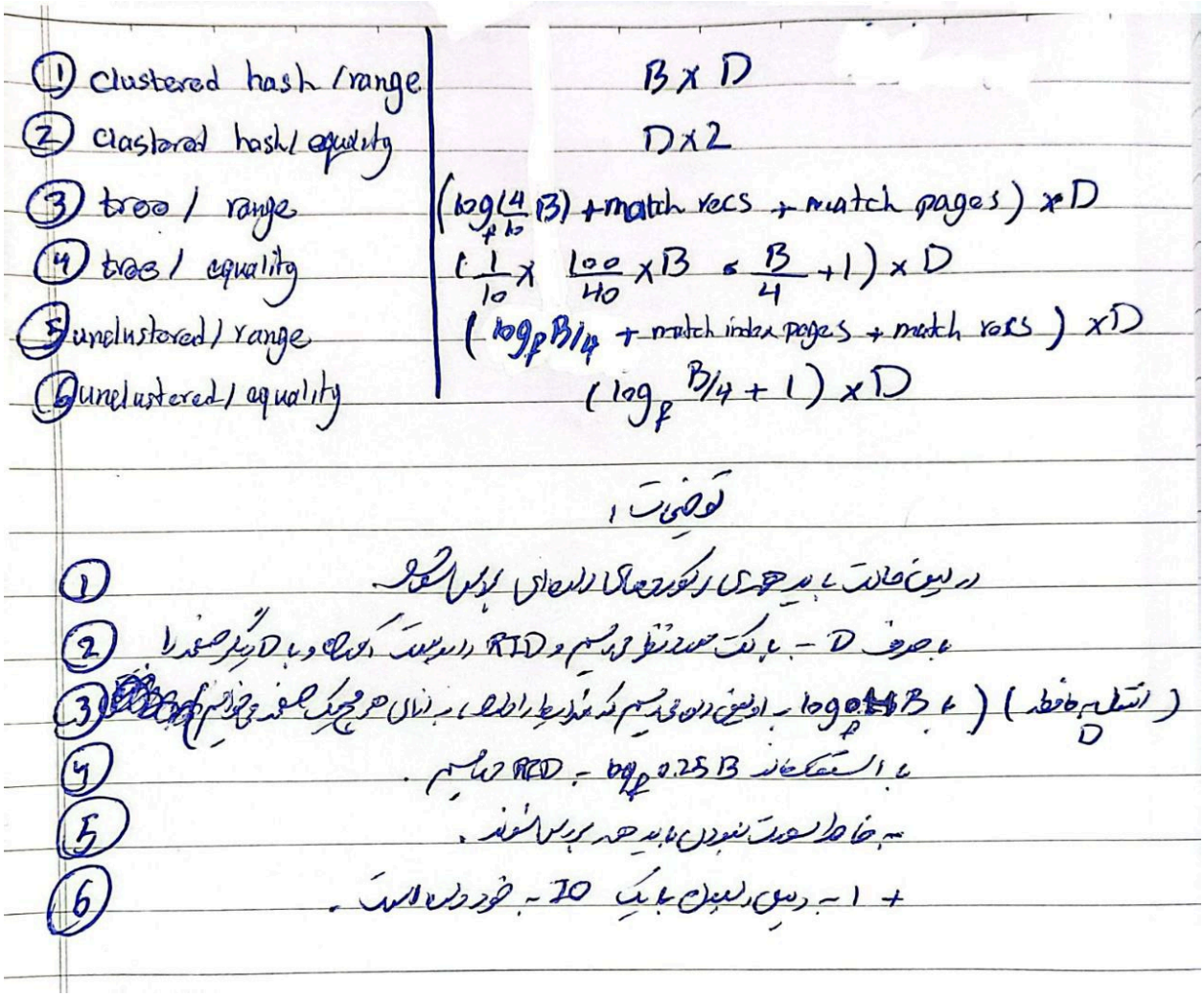
سوال پنجم) روش اول استفاده از delimiter است. براي ذخيره سازي رکوردهايي با طول متغير استفاده از يك جدا کننده (مثل ويرگول) می تواند مورد استفاده باشد که دو مشکل اصلی دارد :
-۱ اسکن کردن براي دسترسي به فيلدها زمان بر است.
-۲ از خود جدا کننده در متن نميتوان استفاده کرد.
روش دوم استفاده از offset است. به طور کلی استفاده از offset برای حل مشکل ذخیره سازی فیلدها با طول متغیر مطرح میشود برای بهبود کارایی استفاده از Delimiter بین فیلدها. چون روش استفاده از Delimiter در صورت به کار رفتن آن کاراکتر در مقادیر فیلدها با مشکل روبرو میشود. حال اگر از Offset استفاده میکنیم، بهتر است اول فیلدها با طول ثابت را قرار دهیم چون میتوان با استفاده از طول آنها (که در کاتالوگ رابطه در DBMS این اطلاعات را نگهداری میکند)، قسمت طول ثابت را skip کرد و اسکن نکرد، و بدین ترتیب نیاز نیست کل رکورد را اسکن کنیم. فقط قسمت با طول متغیر را اسکن میکنیم. در این روش همچنان برای دسترسی به فیلدهای با طول متغیر الزم است کل رکورد اسکن شود که در مقایسه با روش نگه داری header ، زمان زیادی میگیرد. همچنین می توانیم برای هر رکورد، یک header در نظر میگیریم و در آن pointer هایی به انتهای فیلدهای متغیر قرار میدهیم. فیلدهای متغیر را نیز در انتهای رکورد ذخیره میکنیم. بدین شکل دیگر نیازی به اسکن کردن برای دسترسی به فیلدها نداریم. همچنین با این روش، مقادیر NULL نیز به خوبی مدیریت میشوند.



سوال ششم)

page	MRU	hit	LRU	hit
1	1****	0	1****	0
3	13***	0	13***	0
2	132**	0	132**	0
5	1325*	0	1325*	0
6	13256	0	13256	0
4	13254	0	43256	0
5	13254	1	43256	1
3	13254	1	43256	1
2	13254	1	43256	1
1	13254	1	43256	0
6	63254	0	63251	0
1	13254	0	63251	1
2	13254	1	63251	1
1	13254	1	63251	1
3	13254	1	63251	1
2	13254	1	63251	1
4	13254	1	63241	0
1	13254	1	63241	1
5	13254	1	63245	0
		hit rate= 11/19		hit rate = 9/19

سوال هفتم)



سوال هشتم) 1. در سطح ،0 داده strip میشود و نگهداری میشود. واحد strip کردن بلاک است و اطلاعات اضافی نگهداری نمیشود. پس قابلیت اطمینان ندارد و احتمال از دست دادن اطلاعات را داریم با اینکه بازدهی نوشتن خوبی دارد. سطح 0+1 یک کپی به سطح 0 اضافه میکند و مشکل عدم اطمینان در سطح 0+1 را برطرف میکند چون دیسکی که از بین برود قابل جایگزینی است. در این سطح، داده ها در سطح بلاک strip میشوند. از طرف دیگر، خواندن موازی با کارایی بالاتری انجام میشود ولی نوشتن کمی طولانی تر است چون دو تا کپی از هر بلاک موجود است. در سطح ،2 واحد strip کردن بیت است و برای تصحیح خطا از Code Hamming استفاده میشود که به کمک آن، محل بروز خطا را نیز میتوانیم تشخیص دهیم. این سطح برای درخواستهای کوچک مناسب نیست. سطح 3 یک بیت parity برای بازیابی داده

استفاده میکند و حجم اطلاعات افزونه در سطح 2 را کاهش میدهد و مزیت آن نسبت به سطح ،2 استفاده از فضای کمتر است چون پیدا کردن محل failure را از طریق اطلاعات افزونه انجام نمیدهد. این سطح نیز داده را در سطح بیت strip میکند. درباره سطح 4 و 5 نیز در هر دو واحد striping يك بلاك است و از parity استفاده میکنند. درخواست هاي خواندن بر روي يك بلاك با يك O/I انجام میشود و براي نوشتن نیز ۲ دیسك درگیر میشوند. (بلاك و parity). در raid 4 همه parity ها بر روي يك ديسك هستند و اين ديسك ممكن است گلوگاه شود اما در raid 5 بلاكهاي parity پخش شدهاند و درخواستهاي نوشتن ميتواند به صورت همزمان انجام شوند.

به طور خلاصه:

RAID 0 هیچ افزونگی داده ای ارائه نمی دهد. سریع ترین سطح RAID است. اما در صورت خرابی هر یک از درایوها، تمام داده ها از بین می روند.

RAID 0 + 1 از پارتیشن بندی استفاده می کند، کپی از داده نگهداری می کند.در صورت خرابی یک درایو، درایو دیگر می تواند داده ها را بازسازی کند. عمل read و write را موازی انجام می دهد.

RAID 1 از آینه سازی داده ها در دو درایو برای افزونگی استفاده می کند. در صورت خرابی یک درایو، درایو دیگر می تواند داده ها را بازسازی کند. فقط نیمی از فضای ذخیره سازی قابل استفاده است. در واقع عمل read و write به صورت موازی انجام می شود.

RAID 2 از کدگذاری همینگ برای محافظت از داده ها در برابر خرابی درایو استفاده می کند. یعنی error correction یا کد تصحیح خطا دارد. می تواند از خرابی دو درایو بازیابی شود. اما کندتر از RAID 0 و RAID 1 است.

RAID 3 از کدگذاری همینگ و یک درایو اختصاصی برای ذخیره اطلاعات بیت پریتی استفاده می کند. می تواند از خرابی هر یک از درایوهای داده بازیابی شود. اما کندتر از RAID 0 و RAID 1 است و به یک درایو اختصاصی نیاز دارد. Read و write همه دیسک ها را شامل می شود. یک دیسک تنها برای پریتی داریم.

RAID 4 از کدگذاری همینگ برای محافظت از داده ها در برابر خرابی درایو استفاده می کند و از پارتیشن بندی بلوکی برای بهبود عملکرد استفاده می کند. می تواند از خرابی هر یک از درایوهای داده بازیابی شود و اما کندتر از RAID 0 و RAID 1 است. interleaved parity داریم و یک

دیسک بلاک و یک دیسک چک داریم.
RAID 5 از کدگذاری دیسک بلوکی توزیع شده (RAID-DP) برای محافظت از داده ها در برابر خرابی درایو استفاده می کند. می تواند از خرابی هر یک از درایوهای داده بازیابی شود، از پارتیشن بندی بلوکی برای بهبود عملکرد استفاده می کند و از RAID 3 و RAID 4 کارآمدتر است. اما کندتر از RAID 0 و RAID 1 است. همچنین Distributed Parity استفاده میشود. در واقع بلاک های پریتی distribute میشوند و دیسک پریتی باعث ایجاد گلوگاه نمی شود.
RAID 0 برای برنامه هایی که به سرعت بالا نیاز دارند، مانند ویرایش ویدیو، مناسب است.
RAID 1 برای برنامه هایی که به قابلیت اطمینان بالا نیاز دارند، مانند ذخیره سازی داده های مهم، مناسب است.
RAID 2 و RAID 3 به دلیل کندی و پیچیدگی، کمتر رایج هستند.
RAID 4 و RAID 5 برای برنامه هایی که به تعادل بین عملکرد و قابلیت اطمینان نیاز دارند، مانند ذخیره سازی داده های عمومی، مناسب هستند.

.2